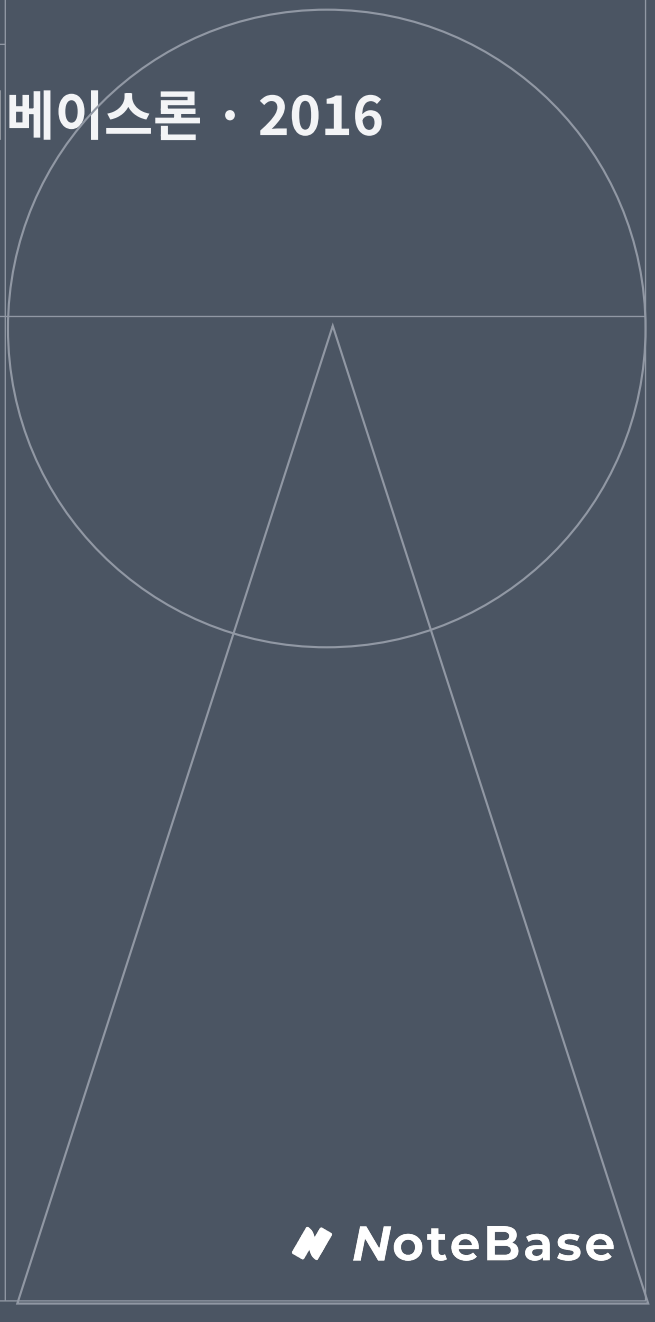
기출OX 해설집

공무원/국가직/7급 · 데이터베이스론 · 2016



NoteBase

기출OX 해설집

공무원/국가직/7급 · 데이터베이스론 · 2016

공무원/국가직/7급 · 데이터베이스론 · 2016

문 1. 다음은 데이터베이스 안에 있는 어떤 테이블을 보인 것이다. 파일처리 방식과 대비해서 <보기>와 같은 테이블을 이용하는 데이터베이스 방식의 주요 특성으로 옳은 것은?

<보 기>

TABLES

Table Name	No of Columns
STUDENT	4
...	...

COLUMNS

Column Name	Data Type	Table Name
StudentNumber	Character(4)	STUDENT
...	...	...

① 데이터 구조가 변경되면 프로그램도 수정해야 하는 성질

② 데이터 중복이 증가해도 일관성을 유지하는 성질

③ 데이터 정의가 응용 프로그램의 일부로 표현되는 성질

④ 데이터베이스 시스템의 스스로를 기술하는 성질

정답 4번

출제 포인트

이 문제는 파일처리 방식과 데이터베이스 방식의 차이 중 시스템 카탈로그를 통한 자기 기술성(self-describing nature)을 묻는 문제이다.

NoteBase

기출OX 해설집 · 공무원/국가직/7급 · 데이터베이스론 · 2016

핵심 개념

자기 기술성: 데이터베이스가 테이블명, 컬럼명, 자료형 같은 메타데이터를 시스템 카탈로그에 저장하여 자기 구조를 스스로 설명하는 성질이다.
프로그램-데이터 독립성: 데이터 구조가 바뀌어도 응용 프로그램 수정 부담을 줄이는 성질이다.
중복 최소화와 일관성: 데이터 중복을 줄여 갱신 이상을 막고 일관성을 높이는 성질이다.

선지별 해설

① X
데이터 구조가 변경되면 프로그램도 수정해야 한다는 내용은 파일처리 방식의 프로그램-데이터 종속성에 해당한다.
틀린 부분은 "프로그램도 수정해야 하는"이다.
올바르게는 데이터베이스 방식은 데이터 독립성이 높아 구조 변경 시 프로그램 수정 부담을 줄이는 성질이다.

② X
데이터 중복이 증가하면 같은 사실을 여러 곳에서 갱신해야 하므로 오히려 불일치가 생기기 쉬워진다.
틀린 부분은 "데이터 중복이 증가해도 일관성을 유지"이다.
올바르게는 데이터베이스 방식은 데이터 중복을 최소화하여 일관성을 유지하는 성질이다.

③ X
데이터 정의가 응용 프로그램의 일부로 표현되는 것은 파일처리 방식의 특징이다.
틀린 부분은 "응용 프로그램의 일부로 표현"이다.
올바르게는 데이터 정의는 응용 프로그램과 분리되어 스키마와 시스템 카탈로그에 저장된다.

④ O
보기의 TABLES, COLUMNS 표는 테이블명, 컬럼명, 자료형처럼 데이터베이스 자신의 구조 정보를 저장한 메타데이터이다.
데이터베이스 시스템이 자기 자신에 대한 정보를 내부 테이블로 기술하므로 자기 기술성의 설명으로 옳다.

오답 포인트

①과 ③은 모두 파일처리 방식의 종속성을 데이터베이스 특성처럼 바꿔 놓은 선지라서 헷갈리기 쉽다. ②는 중복과 일관성의 관계를 반대로 제시한 함정이다. 보기의 표가 사용자 데이터가 아니라 테이블과 컬럼을 설명하는 메타데이터라는 점을 놓치면 ④를 고르기 어렵다.

NoteBase

다시 나오면 이렇게 판단

그림이나 설명에 테이블명, 컬럼명, 자료형처럼 데이터에 대한 데이터가 보이면 먼저 시스템 카탈로그와 자기 기술성을 떠올린다. 선지에서 구조 변경 시 프로그램 수정, 응용 프로그램 내부 정의, 중복 증가 같은 표현이 나오면 데이터베이스의 장점이 아니라 파일처리 방식의 한계 또는 중복 문제로 판단한다.

암기 포인트

테이블을 설명하는 테이블은 자기 기술성, 구조 변경에도 프로그램 수정 부담을 줄이는 것은 데이터 독립성이다.

NoteBase

문 2. 하나의 데이터베이스 시스템 내에서 적절한 제어 없이 트랜잭션들을 동시에 실행하였을 경우 여러 문제가 발생할 수 있다. 이를 해결하기 위한 동시성 제어가 올바르게 동작하지 않을 경우 발생할 수 있는 문제점으로 옳지 않은 것은?

① 갱신 손실 문제

② 부정확한 요약 문제

③ 반복할 수 없는 읽기 문제

④ 지역적 오류 문제

정답 4번

출제 포인트

이 문제는 동시성 제어가 실패할 때 나타나는 대표적 이상 현상과 비표준 용어를 구분할 수 있는지를 묻는 문제이다.

핵심 개념

갱신 손실은 둘 이상의 트랜잭션이 같은 데이터를 갱신하는 과정에서 한쪽 갱신 결과가 다른 쪽에 덮여 사라지는 현상이다.
부정확한 요약은 합계나 평균 같은 집계 연산이 갱신 중인 데이터 일부를 섞어 읽어 어느 한 시점과도 맞지 않는 요약값을 만드는 현상이다.
반복할 수 없는 읽기는 같은 트랜잭션이 같은 데이터를 두 번 읽었을 때 중간에 다른 트랜잭션의 갱신이 반영되어 값이 달라지는 현상이다.
지역적 오류는 동시성 제어의 대표적 이상 현상을 가리키는 표준 용어로 보지 않는다.

NoteBase

선지별 해설

① O
갱신 손실 문제는 동시 실행된 트랜잭션이 동일 데이터 항목을 각각 수정할 때 발생하는 대표적 동시성 이상 현상이다. 한 트랜잭션의 갱신 결과가 다른 트랜잭션의 기록으로 덮이면 먼저 한 수정이 사라진다.

② O
부정확한 요약 문제는 집계 중간에 다른 트랜잭션의 갱신이 끼어들어 합계나 평균이 비일관적으로 계산되는 현상이다. 동시성 제어가 제대로 되지 않을 때 발생할 수 있는 전형적 문제이므로 옳다.

③ O
반복할 수 없는 읽기 문제는 같은 트랜잭션 내부에서 동일 데이터를 다시 읽었을 때 값이 달라지는 현상이다. 이는 다른 트랜잭션의 갱신이 중간에 개입한 결과이므로 동시성 제어 실패와 관련된 대표 문제이다.

④ X
틀린 부분은 "지역적 오류 문제"라는 표현이다. 이는 동시성 제어의 대표적 이상 현상을 가리키는 표준 용어가 아니다.
올바르게는 "동시성 제어가 실패할 때는 오손 읽기 문제, 갱신 손실 문제, 부정확한 요약 문제, 반복할 수 없는 읽기 문제 등이 발생할 수 있다"라고 이해하는 편이 맞다.

오답 포인트

①, ②, ③은 데이터베이스 동시성 이상 현상으로 자주 함께 묶여 출제되므로 익숙한 용어이다. ④는 실제 개념처럼 보이는 낯선 표현을 끼워 넣어 용어 암기만 한 수험생이 속기 쉬운 선택지이다.

다시 나오면 이렇게 판단

동시성 제어 문항에서는 먼저 각 보기의 용어가 실제 이상 현상 명칭인지부터 확인한다. 그다음 같은 데이터의 덮어쓰기이면 갱신 손실, 집계값 왜곡이면 부정확한 요약, 같은 항목 재조회 값 변화이면 반복할 수 없는 읽기로 연결하면 된다.

암기 포인트

동시성 이상 현상은 덮어쓰기면 갱신 손실, 집계 왜곡이면 부정확한 요약, 재조회 값 변화면 반복할 수 없는 읽기이다.

NoteBase

문 3. 다음 데이터를 순서적으로 삽입하여 B-트리를 구성할 때 루트 노드에 존재하는 키 값은? (단, B-트리의 차수는 3이라고 가정한다)

키 값들의 삽입순서: 8, 5, 1, 7, 3, 12, 9, 6

① 3

② 5

③ 7

④ 9

정답 2번

출제 포인트

이 문제는 차수 3인 B-트리에 키를 순서대로 삽입할 때 노드 분할이 어떻게 일어나고 최종 루트에 어떤 키가 남는지를 묻는 문제이다.

올바른 순서

1. 8, 5를 삽입하면 루트는 [5, 8]이 된다.
2. 1을 삽입하면 [1, 5, 8]이 되어 overflow가 발생하므로 가운데 키 5가 루트로 승격되고 자식은 [1], [8]이 된다.
3. 7, 3을 차례로 삽입하면 왼쪽 자식은 [1, 3], 오른쪽 자식은 [7, 8]이 된다.
4. 12를 삽입하면 오른쪽 자식 [7, 8, 12]가 분할되어 8이 루트로 올라가고 루트는 [5, 8], 자식은 [1, 3], [7], [12]가 된다.
5. 9, 6을 삽입하면 각 자식은 [1, 3], [6, 7], [9, 12]가 되고 추가 분할이 없으므로 최종 루트 키는 5와 8이다.

왜 이 순서: 차수 3 B-트리는 한 노드에 키를 최대 2개만 둘 수 있으므로 세 번째 키가 들어가는 순간마다 가운데 키가 부모로 승격된다.

NoteBase

선지별 해설

① X
3은 최종 루트 키가 아니라 왼쪽 리프 노드 [1, 3]에 남는다. 틀린 표현은 '루트 노드의 키가 3이다'이고, 올바르게는 '3은 왼쪽 리프 노드에 있고 루트 키는 5와 8이다'이다.

② O
5는 1을 삽입하면서 첫 분할이 일어날 때 가운데 키로 승격되어 루트에 올라가고, 이후 추가 삽입 뒤에도 루트에 그대로 남는다.

③ X
7은 12 삽입 후 분할이 끝난 뒤 가운데 자식 [7]에 있다가 6이 들어오면 [6, 7]이 될 뿐 루트로 승격되지 않는다. 틀린 표현은 '루트 노드의 키가 7이다'이고, 올바르게는 '7은 가운데 자식 노드에 있고 루트 키는 5와 8이다'이다.

④ X
9는 마지막에 오른쪽 자식 [9, 12]에 들어가므로 루트 키가 아니다. 틀린 표현은 '루트 노드의 키가 9이다'이고, 올바르게는 '9는 오른쪽 리프 노드에 있고 루트 키는 5와 8이다'이다.

오답 포인트

분할이 일어난 직후의 승격 키만 보고 마지막 루트 구조를 끝까지 추적하지 않으면 3, 7, 9처럼 리프나 중간 자식에 남는 값을 루트 키로 착각하기 쉽다. 특히 차수 3에서는 키가 3개가 되는 순간 가운데 값만 부모로 올라간다는 규칙을 놓치기 쉽다.

다시 나오면 이렇게 판단

B-트리 삽입 문제는 각 단계마다 현재 노드의 키 개수를 확인하고, 키가 최대 개수를 넘는 순간 가운데 키가 부모로 승격되는지만 추적하면 된다. 마지막까지 승격된 키들만 루트에 남고 나머지는 해당 자식 노드에 남는다.

암기 포인트

차수 3 B-트리는 키 3개가 되는 순간 가운데 키 1개가 위로 올라간다고 기억하면 된다.

NoteBase

문 4. 두 릴레이션 R1(A, B, C), R2(B, C, D)를 오른쪽 외부조인(Right Outer Join)을 한 결과에 나타나는 튜플의 수는?

R1		
A	B	C
1	2	3
4	2	3
7	8	9

R2		
B	C	D
2	3	10
2	3	11
6	7	12

① 3개

② 4개

③ 5개

④ 6개

정답 3번

출제 포인트

이 문제는 오른쪽 외부조인에서 오른쪽 릴레이션의 보존 여부와 조인 속성의 다중 매칭으로 늘어나는 튜플 수를 계산하는 문제이다.

조건 정리

조인에 공통으로 쓰이는 속성은 B, C이다.
R1에서 (B,C)=(2,3)인 튜플은 2개이고, R2에서 (B,C)=(2,3)인 튜플도 2개이다.
오른쪽 외부조인은 R2의 모든 튜플을 남기므로 R2의 미매칭 튜플도 결과에 포함된다.

NoteBase

계산 과정

R2의 첫 번째 튜플 (2,3,10)은 R1의 두 튜플과 모두 조인되어 2개 결과를 만든다.
R2의 두 번째 튜플 (2,3,11)도 R1의 두 튜플과 모두 조인되어 2개 결과를 만든다.
R2의 세 번째 튜플 (6,7,12)은 R1에서 일치하는 (B,C)가 없으므로 오른쪽 외부조인에 따라 1개 결과가 그대로 남는다.
따라서 전체 튜플 수는 2 + 2 + 1 = 5개이다.

선지별 해설

① X: 3개는 틀리다. R2의 튜플 수만 그대로 세면 3개가 되지만, (B,C)=(2,3)에서 2대2 매칭이 일어나므로 결과는 4개가 아니라 4개 매칭 결과에 미매칭 1개를 더한 5개이다. 틀린 표현은 '3개'이고, 올바르게는 '5개'이다.
② X: 4개는 틀리다. (2,3) 매칭으로 생기는 4개까지만 세고, R2의 미매칭 튜플 (6,7,12)이 오른쪽 외부조인에서 보존된다는 점을 빼먹은 값이다. 틀린 표현은 '4개'이고, 올바르게는 '5개'이다.
③ O: 5개는 옳다. (2,3) 매칭에서 4개가 나오고, R2의 미매칭 튜플 1개가 추가되어 총 5개가 된다.
④ X: 6개는 틀리다. 왼쪽 릴레이션 R1의 미매칭 튜플 (7,8,9)까지 보존된다고 착각하면 과대계산이 된다. 오른쪽 외부조인은 오른쪽인 R2만 보존하므로 틀린 표현은 '6개'이고, 올바르게는 '5개'이다.

오답 포인트

오른쪽 외부조인을 단순히 '조인 후 없는 값은 NULL로 채운다' 정도로만 기억하면 어느 쪽 릴레이션이 보존되는지 놓치기 쉽다. 또 (B,C)=(2,3)처럼 동일한 조인값이 여러 행에 반복될 때는 1대1이 아니라 매칭 수의 곱으로 결과 튜플 수가 늘어난다는 점을 놓치기 쉽다.

다시 나오면 이렇게 판단

먼저 조인 기준 속성이 무엇인지 확인하고, 같은 조인값끼리 몇 대 몇으로 매칭되는지 센다. 그다음 외부조인에서 보존되는 쪽이 왼쪽인지 오른쪽인지 확인한 뒤, 보존되는 릴레이션의 미매칭 튜플만 추가하면 된다.

암기 포인트

외부조인 튜플 수는 '매칭 결과 수 + 보존되는 쪽의 미매칭 수'로 계산한다.

NoteBase

문 5. 다음 두 릴레이션 R, S에 대한 조인 결과로 옳지 않은 것은? (단, ⋈ $_N$은 자연조인(Natural Join), ⟕은 왼쪽 외부조인(Left Outer Join), ⟖은 오른쪽 외부 조인(Right Outer Join)을 의미한다)

R

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4

S

B	C	D
b1	c1	d1
b1	c2	d2
b3	c3	d3
b4	c4	d4

① $R \bowtie_N S$

A	B	C	D
a1	b1	c1	d1
a3	b3	c3	d3
a4	b4	c4	d4

② $R \bowtie_{b=b \wedge c=c} S$

A	B	C	B	C	D
a1	b1	c1	b1	c1	d1
a2	b1	c2	b1	c2	d2
a3	b3	c3	b3	c3	d3
a4	b4	c4	b4	c4	d4

③ $R \bowtie_{b=b \wedge c=c} S$

A	B	C	B	C	D
a1	b1	c1	b1	c1	d1
a2	b2	c2	NULL	NULL	NULL
a3	b3	c3	b3	c3	d3
a4	b4	c4	b4	c4	d4

NoteBase

④ $R \bowtie_{b=b \wedge c=c} S$

A	B	C	B	C	D
a1	b1	c1	b1	c1	d1
NULL	NULL	NULL	b1	c2	d2
a3	b3	c3	b3	c3	d3
a4	b4	c4	b4	c4	d4

정답 2번

출제 포인트

이 문제는 공통 속성 *B*, *C*를 기준으로 자연조인, 내부조인, 왼쪽 외부조인, 오른쪽 외부조인의 결과 행과 NULL 위치를 정확히 구분하는 문제이다.

자료 읽는 순서

1. R과 S의 공통 속성이 *B*, *C*라는 점을 먼저 확인한다.
2. *B*와 *C*가 동시에 같은 행만 내부조인과 자연조인에 남는다는 점을 확인한다.
3. 외부조인은 어느 쪽 릴레이션을 보존하는지 먼저 정한 뒤, 일치하지 않는 행에 NULL이 어느 쪽에 들어가는지 대조한다.

선지별 해설

① O: 자연조인은 공통 속성 *B*, *C*가 모두 같은 행만 남기고 공통 속성 열을 한 번만 표시한다. R의 1행 ($a1, b1, c1$), 3행 ($a3, b3, c3$), 4행 ($a4, b4, c4$)만 S의 각 대응 행과 모두 일치하므로 결과가 제시된 3행이 맞다.

② X: 조건조인 R _b=b c=c S는 *B*와 *C*가 동시에 일치하는 행만 남긴다. 그런데 보기의 2행은 R의 2행 ($a2, b2, c2$)와 S의 2행 ($b1, c2, d2$)를 붙였으므로 *C*만 같고 *B*는 $b2 \neq b1$로 다르다. 틀린 표현은 이 2행이 결과에 포함된다는 점이다. 올바르게는 이 행이 빠져야 하며, 전체 결과는 1행, 3행, 4행의 3개 행만 남는다.

③ O: 왼쪽 외부조인은 R의 모든 행을 보존한다. 따라서 일치하지 않는 R의 2행 ($a2, b2, c2$)도 남고, 대응하는 S 쪽 속성 *B*, *C*, *D*에는 NULL이 채워진다. 제시된 결과가 맞다.

④ O: 오른쪽 외부조인은 S의 모든 행을 보존한다. S의 2행 ($b1, c2, d2$)은 R에 *B*, *C*가 동시에 일치하는 행이 없으므로, R 쪽 속성 *A*, *B*, *C*가 NULL로 표시되고 S 쪽 값은 유지된다. 제시된 결과가 맞다.

오답 포인트

②는 *C* 값이 $c2$로 같다는 이유만으로 조인된 것처럼 보이게 만든 함정이다. 이 문제의 조인 조건은 *B*와 *C*를 동시에 만족해야 하므로, 한 속성만 같으면 결과에 포함되지 않는다. 또 외부조인은 보존되는 쪽을 먼저 정하지 않으면 NULL 위치를 반대로 읽기 쉽다.

NoteBase

다시 나오면 이렇게 판단

먼저 조인 조건에 포함된 속성이 무엇인지 표시한 뒤, 각 행이 그 조건을 모두 만족하는지 확인한다. 그다음 자연조인과 내부조인은 일치 행만 남긴다고 보고, 외부조인은 보존되는 릴레이션의 모든 행을 남긴 뒤 반대쪽에만 NULL이 들어가는지 판단한다.

암기 포인트

내부조인은 일치 행만 남고, 왼쪽 외부조인은 왼쪽을, 오른쪽 외부조인은 오른쪽을 끝까지 보존한다.

NoteBase

문 6. 도서(도서번호, 도서제목, 출판사명, 발행년도) 테이블에서, 2000년 이후에 10권 이상의 책을 발행한 출판사의 이름을 중복 없이 출력하는 SQL문으로 옳은 것은? (단, 출판사명이 동일한 출판사는 존재하지 않는 것으로 가정한다. 도서번호는 도서 테이블의 기본키이다)

①
```
SELECT 출판사명 FROM 도서
WHERE 발행년도 >= 2000
ORDER BY COUNT(도서번호) >= 10;
```

②
```
SELECT 출판사명 FROM 도서
WHERE 발행년도 >= 2000 AND COUNT(도서번호) >= 10
GROUP BY 출판사명;
```

③
```
SELECT 출판사명 FROM 도서
WHERE COUNT(도서번호) >= 10
GROUP BY 출판사명 HAVING 발행년도 >= 2000;
```

④
```
SELECT 출판사명 FROM 도서
WHERE 발행년도 >= 2000
GROUP BY 출판사명 HAVING COUNT(도서번호) >= 10;
```

정답 4번

출제 포인트

이 문제는 행을 먼저 거르는 WHERE와 그룹을 만든 뒤 그룹별 조건을 거는 HAVING의 차이를 묻는 문제이다.

핵심 개념

WHERE는 GROUP BY 이전에 개별 행을 거르는 조건절이다.
GROUP BY는 같은 출판사명끼리 행을 묶어 그룹을 만드는 절이다.
HAVING은 GROUP BY 이후에 만들어진 그룹에 대해 COUNT 같은 집계조건을 거는 절이다.
이 문제에서는 먼저 발행년도 조건으로 책 행을 걸러야 하고, 그다음 출판사별로 묶은 뒤 권수 조건을 판단해야 한다.

NoteBase

선지별 해설

① X
ORDER BY는 정렬 절이지 10권 이상인 출판사를 걸러내는 절이 아니다. 또한 GROUP BY 없이 COUNT(도서번호)를 정렬 기준처럼 쓴 것도 맞지 않는다.
틀린 표현은 "ORDER BY COUNT(도서번호) >= 10"이다. 올바르게는 GROUP BY 출판사명 HAVING COUNT(도서번호) >= 10 형태로 그룹 조건을 걸어야 한다.

② X
WHERE는 개별 행을 거르는 곳이므로 COUNT(도서번호) 같은 집계함수를 둘 수 없다.
틀린 표현은 "WHERE 발행년도 >= 2000 AND COUNT(도서번호) >= 10"이다. 올바르게는 발행년도 조건은 WHERE에 두고, COUNT(도서번호) >= 10은 HAVING에 둔다.

③ X
COUNT(도서번호) >= 10을 WHERE에 둔 점이 틀렸고, 발행년도 >= 2000을 HAVING에 둔 점도 부적절하다. 발행년도는 각 행의 속성이므로 그룹화 전에 먼저 걸러야 한다.
틀린 표현은 "WHERE COUNT(도서번호) >= 10"과 "HAVING 발행년도 >= 2000"이다. 올바르게는 WHERE 발행년도 >= 2000 GROUP BY 출판사명 HAVING COUNT(도서번호) >= 10이다.

④ O
발행년도 >= 2000인 책들만 먼저 WHERE로 선택한 뒤, 출판사명으로 GROUP BY 하고, 각 출판사 그룹의 책 수가 10권 이상인 경우만 HAVING으로 남긴 형태이다. SELECT 출판사명은 그룹당 한 번씩만 나오므로 요구한 출판사 이름이 중복 없이 출력된다.

오답 포인트

출제자는 COUNT 조건이 보이면 무조건 WHERE에 넣거나, HAVING을 단순한 두 번째 WHERE처럼 오해하는 지점을 노린다. 이 문항에서는 "발행년도"는 행 조건이고 "10권 이상"은 그룹 조건이라는 구분만 잡히면 선지가 바로 갈린다.

다시 나오면 이렇게 판단

조건이 개별 행의 값인지, 그룹에 대한 집계결과인지부터 구분하면 된다. 행 자체의 속성을 제한하면 WHERE이고, COUNT·SUM·AVG처럼 묶인 결과를 제한하면 HAVING이다.

암기 포인트

WHERE는 행, HAVING은 그룹이다. 집계함수 조건은 HAVING에 둔다.

NoteBase

문 7. 지연 갱신을 통한 점진적 로깅(incremental logging with deferred updates) 기법을 사용하는 복구 시스템으로 옳은 것은?

① 갱신된 데이터 아이템의 갱신 전 원래 값만을 로그에 저장한다.

② 갱신된 데이터 아이템의 갱신 후 수정된 값만을 로그에 저장한다.

③ 갱신된 데이터 아이템의 갱신 전, 후 값을 모두 로그에 저장해야 한다.

④ Begin Transaction과 Commit Transaction 레코드만을 로그에 저장하면 된다.

정답 2번

출제 포인트

이 문제는 지연 갱신을 사용하는 deferred update 복구에서 로그에 어떤 값을 남기는지 묻는 문제이다.

핵심 개념

지연 갱신은 트랜잭션이 커밋되기 전에는 데이터베이스의 실제 항목을 바로 갱신하지 않고, 로그에 변경 내용을 기록해 두었다가 커밋 후 필요 시 재실행으로 반영하는 방식이다.
따라서 복구의 핵심은 갱신 후 값(after image)이며, 갱신 전 값(before image)은 기본 전제가 아니다.

선지별 해설

① X
갱신 전 원래 값만 저장하는 방식은 before image 중심 설명이다. 지연 갱신은 커밋 전 실제 데이터베이스 반영을 미루므로 redo를 위해 after image가 필요하다. 틀린 표현은 "갱신 전 원래 값만"이다. 올바르게는 갱신 후 수정된 값이 로그에 저장된다.

② O
지연 갱신을 통한 점진적 로깅은 변경된 데이터 항목의 갱신 후 값(after image)을 로그에 저장하고, 장애 시 그 값을 다시 적용하는 방식이다. 따라서 2번이 옳다.

③ X
갱신 전후 값을 모두 저장하는 것은 즉시 갱신에서 undo와 redo를 함께 고려할 때의 설명에 가깝다. 틀린 표현은 "갱신 전, 후 값을 모두"이다. 올바르게는 지연 갱신에서는 보통 갱신 후 값만 기록한다.

④ X
Begin Transaction과 Commit Transaction 레코드만으로는 어떤 데이터 항목이 어떻게 바뀌었는지 알 수 없어 재실행할 수 없다. 틀린 표현은 "레코드만"이다. 올바르게는 트랜잭션 경계 정보와 함께 갱신된 데이터 항목의 after image도 로그에 저장해야 한다.

NoteBase

오답 포인트

지연 갱신과 즉시 갱신을 섞어 기억하면 before image, after image, before/after image 모두 저장이라는 문장이 뒤섞인다. 특히 ③은 정보가 더 많아 보인다는 이유로 고르기 쉽지만, 지연 갱신의 핵심은 실제 반영을 늦추고 redo 중심으로 복구한다는 점이다.

다시 나오면 이렇게 판단

복구 기법 문제는 먼저 커밋 전 실제 데이터베이스를 갱신하는지부터 본다. 커밋 전 반영하지 않으면 지연 갱신이므로 redo 중심, after image 중심으로 판단한다. 커밋 전 반영하면 즉시 갱신이므로 undo 필요성을 함께 판단한다.

암기 포인트

지연 갱신은 나중에 반영하므로 redo, 즉시 갱신은 미리 반영하므로 undo까지 고려하는 방식으로 정리된다.

NoteBase

문 8. 데이터베이스에서 동시성 제어가 적절하게 이루어지지 않으면서 다음과 같이 두 트랜잭션 T1, T2가 동시에 실행될 때 문제점이 발생할 수 있다. 이 문제점이 발생하지 않을 수 있는 트랜잭션의 격리 수준(isolation level)을 모두 고른 것은?

T1	T2
update product set amount = amount + new where pName = “chair” ;	
	select productID, amount from product ;
rollback ;	
	commit ;

① 미완료 읽기, 완료 읽기, 반복가능 읽기, 직렬 가능

② 완료 읽기, 반복가능 읽기, 직렬 가능

③ 반복가능 읽기, 직렬 가능

④ 직렬 가능

정답 2번

출제 포인트

이 문제는 롤백될 미완료 갱신을 다른 트랜잭션이 읽는 더티 리드를 막을 수 있는 격리 수준을 묻는 문제이다.

사례 정리

T1은 chair 행의 amount를 갱신한 뒤 커밋하지 않은 상태이다.
그 사이 T2가 product 테이블을 조회하여 T1의 변경값을 읽을 수 있다.
이후 T1이 rollback하므로 T2가 읽은 값은 실제로는 취소될 수 있는 값이라는 점이 판단의 핵심이다.

NoteBase

핵심 개념

미완료 읽기(Read Uncommitted)는 다른 트랜잭션의 커밋 전 변경을 읽을 수 있는 격리 수준이다.
완료 읽기(Read Committed) 이상에서는 커밋된 데이터만 읽으므로 더티 리드를 방지한다.
반복가능 읽기와 직렬 가능도 완료 읽기보다 강한 수준이므로 더티 리드를 함께 방지한다.

적용 기준

이 사례의 문제점은 T2가 T1의 커밋 전 갱신값을 읽고, 그 뒤 T1이 롤백하는 상황이라는 점이다.
따라서 판단 기준은 더티 리드를 허용하는지 여부이다.
더티 리드를 허용하는 미완료 읽기는 제외하고, 이를 허용하지 않는 완료 읽기 이상만 고르면 된다.

선지별 해설

① X
미완료 읽기는 커밋되지 않은 값을 읽을 수 있으므로 이 사례의 문제점을 막지 못한다.
틀린 표현은 '미완료 읽기'이다. 올바르게는 '완료 읽기, 반복가능 읽기, 직렬 가능'이어야 한다.

② O
완료 읽기는 더티 리드를 막고, 반복가능 읽기와 직렬 가능은 그보다 더 강한 격리 수준이므로 역시 이 문제를 막을 수 있다.
주어진 사례에서 필요한 수준을 정확히 모두 포함한 선지이다.

③ X
반복가능 읽기와 직렬 가능은 맞지만 완료 읽기도 이미 더티 리드를 방지한다.
빠진 표현은 '완료 읽기'이다. 올바르게는 '완료 읽기, 반복가능 읽기, 직렬 가능'이어야 한다.

④ X
직렬 가능은 물론 가능하지만, 더 낮은 수준인 완료 읽기와 반복가능 읽기도 이미 이 문제를 막을 수 있다.
틀린 표현은 '직렬 가능만 가능하다'는 취지이다. 올바르게는 '완료 읽기, 반복가능 읽기, 직렬 가능이 모두 가능하다'이다.

오답 포인트

반복가능 읽기나 직렬 가능처럼 강한 격리 수준만 떠올리고, 완료 읽기도 더티 리드를 막는다는 기본 구분을 놓치기 쉽다. 이 문제는 비반복 읽기나 팬텀 리드가 아니라 롤백될 값을 읽는 더티 리드 상황이라는 점이 핵심이다.

NoteBase

다시 나오면 이렇게 판단

트랜잭션 사례에서 먼저 커밋 전 데이터를 다른 트랜잭션이 읽었는지를 본다. 읽었다가 원 트랜잭션이 롤백하면 더티 리드이므로, 이를 막는 최소 수준인 완료 읽기부터 정답 후보에 포함하면 된다.

암기 포인트

더티 리드는 미완료 읽기만 허용하고, 완료 읽기 이상은 모두 차단한다.

NoteBase

문 9. 다음 ERD(Entity-Relationship Diagram)를 구성하는 두 릴레이션 Dept, Emp에 대해 참조무결성 제약조건을 정의하였다고 가정하자. 정의한 참조무결성 제약조건의 영향을 받는 연산들로만 모두 묶은 것은?



ㄱ. Emp 테이블에 새로운 레코드를 삽입하는 삽입 연산
ㄴ. Emp 테이블에서 외래키 값을 수정하는 변경 연산
ㄷ. Emp 테이블에서 기존 레코드를 삭제하는 삭제 연산
ㄹ. Dept 테이블에 새로운 레코드를 삽입하는 삽입 연산
ㅁ. Dept 테이블에서 기본키 값을 수정하는 변경 연산
ㅂ. Dept 테이블에서 기존 레코드를 삭제하는 삭제 연산

① ㄱ, ㄷ, ㅁ
② ㄴ, ㄹ, ㅂ
③ ㄷ, ㄹ, ㅁ
④ ㄱ, ㄴ, ㅁ, ㅂ

정답 4번

출제 포인트

이 문제는 ERD의 1:N 관계를 릴레이션으로 변환했을 때 Dept가 부모 테이블이고 Emp가 자식 테이블이라는 점을 바탕으로, 참조무결성 제약의 영향을 받는 연산을 고르는 문제이다.

핵심 개념

참조무결성은 자식 테이블의 외래키가 부모 테이블의 기본키를 올바르게 참조해야 한다는 제약이다. 따라서 자식의 외래키를 넣거나 바꾸는 연산, 부모의 기본키를 바꾸거나 부모 행을 지우는 연산이 직접 영향을 받는다.

NoteBase

선지별 해설

① X: ㄱ은 맞지만 ㄷ이 틀렸다. Emp 테이블에서 기존 레코드를 삭제하는 연산은 자식 행을 없애는 연산이므로 고아 레코드를 만들지 않아 참조무결성 제약의 직접 대상이 아니다. 또한 ㄴ이 빠져 있다. 틀린 조합 → 올바르게는 ㄱ, ㄴ, ㅁ, ㅂ이다.
② X: ㄴ과 ㅂ은 맞지만 ㄹ이 틀렸다. Dept 테이블에 새로운 레코드를 삽입하는 연산은 부모 키 값을 추가하는 것이므로 기존 Emp의 외래키 참조를 깨지 않는다. 또한 ㄱ과 ㅁ이 빠져 있다. 틀린 조합 → 올바르게는 ㄱ, ㄴ, ㅁ, ㅂ이다.
③ X: ㄷ과 ㄹ이 모두 틀렸다. Emp 삭제는 자식 쪽 삭제라 참조무결성 위반을 만들지 않고, Dept 삽입도 기존 참조를 깨지 않는다. 반대로 ㄱ, ㄴ, ㅁ, ㅂ이 영향 대상이다. 틀린 조합 → 올바르게는 ㄱ, ㄴ, ㅁ, ㅂ이다.
④ O: ㄱ Emp 삽입은 외래키가 존재하는 Dept를 참조하는지 검사해야 하고, ㄴ Emp의 외래키 변경도 같은 검사를 받는다. ㅁ Dept의 기본키 변경은 Emp의 참조를 깨뜨릴 수 있고, ㅂ Dept 삭제도 소속 Emp를 고아 레코드로 만들 수 있으므로 참조무결성 제약의 영향을 받는다.

오답 포인트

자식 테이블 삭제와 부모 테이블 삽입을 영향 대상이라고 착각하면 틀리기 쉽다. 참조무결성은 없는 부모를 참조하게 되거나, 이미 참조되는 부모 키가 사라지거나 바뀌는 상황을 막는 제약이므로 자식 삭제와 부모 삽입은 보통 직접 문제가 되지 않는다.

다시 나오면 이렇게 판단

1:N 관계가 보이면 먼저 부모 테이블의 기본키와 자식 테이블의 외래키를 찾는다. 그다음 외래키 값을 넣거나 바꾸는 연산, 그리고 참조되는 기본키를 바꾸거나 지우는 연산만 표시하면 된다.

암기 포인트

참조무결성은 자식 FK 입력·변경, 부모 PK 변경·삭제를 본다.

NoteBase

문 10. 데이터베이스는 다양한 응용을 위해 사용되고 있으며 이를 위해서 새로운 데이터베이스 기술이 등장하였다. 새로운 데이터베이스 기술에 대한 설명으로 옳지 않은 것은?

① 공간 데이터베이스에서 다루는 데이터의 형태는 점, 선, 폴리곤 등을 포함하는 지도데이터, 지도의 특징을 기술하는 속성 데이터, 위성이나 항공사진과 같은 이미지 데이터가 주를 이룬다.

② 공간 데이터에 대한 색인을 위해서 그리드 파일, R-트리와 같은 새로운 색인 기법이 사용된다.

③ 공간조인은 주로 지도상의 점, 선, 폴리곤과 이에 대한 속성 데이터를 조인하는 데에 사용된다.

④ 공간 데이터 마이닝의 주요 기법은 공간 분류, 공간 연관성 분석, 공간 클러스터링이 있으며, 이때 공간 데이터에 대해서는 트랜잭션 개념을 가지고 분할하여 분석하지 않는다.

정답 3번

출제 포인트

이 문제는 공간 데이터베이스의 핵심 개념인 공간 데이터, 공간 색인, 공간조인, 공간 데이터 마이닝의 차이를 구분하는 문제이다.

핵심 개념

공간 데이터베이스는 위치와 형태를 가진 공간 객체를 저장하고 질의하는 데이터베이스이다.
공간 색인은 점, 선, 폴리곤 같은 공간 객체를 빠르게 찾기 위한 구조이며 R-트리, 그리드 파일 등이 대표적이다.
공간조인은 두 공간 객체나 두 공간 릴레이션을 공간 관계(포함, 인접, 교차 등)로 결합하는 연산이다.
공간 데이터 마이닝은 공간 분류, 공간 연관성 분석, 공간 클러스터링처럼 공간적 인접성과 분포 특성을 함께 분석하는 기법이다.

NoteBase

선지별 해설

① O
공간 데이터베이스는 지도데이터, 속성데이터, 위성사진·항공사진 같은 래스터 이미지 데이터를 함께 다룬다. 공간 데이터의 표현을 설명한 내용으로 옳다.

② O
공간 데이터는 일차원 정렬이 어려워 일반 B-트리만으로 처리하기 불편하므로, 그리드 파일이나 R-트리 같은 공간 색인 기법을 사용한다. 공간 검색 성능을 높이기 위한 설명으로 옳다.

③ X
틀린 부분은 '지도상의 점, 선, 폴리곤과 이에 대한 속성데이터를 조인하는 데에 사용된다'이다. 공간조인은 공간 객체와 그 속성값을 단순 결합하는 연산이 아니라, 두 공간 객체 또는 두 공간 릴레이션을 공간적 관계를 기준으로 결합하는 연산이다. 올바르게는 '공간조인은 두 공간 데이터 집합을 포함, 인접, 교차 등의 공간 관계로 조인하는 데 사용된다'이다.

④ O
공간 데이터 마이닝의 주요 기법으로 공간 분류, 공간 연관성 분석, 공간 클러스터링을 들 수 있다. 또한 일반 장바구니 분석처럼 독립적인 트랜잭션 단위로 잘라 보기보다 공간적 연속성과 인접성을 함께 고려하므로 진술이 옳다.

오답 포인트

③은 '조인'이라는 단어 때문에 일반 관계형 조인과 같은 방식으로 착각하기 쉽다. 속성데이터를 붙이는 조인과 공간 관계를 기준으로 결합하는 공간조인을 구분하지 못하면 ③을 옳다고 보기 쉽다.

다시 나오면 이렇게 판단

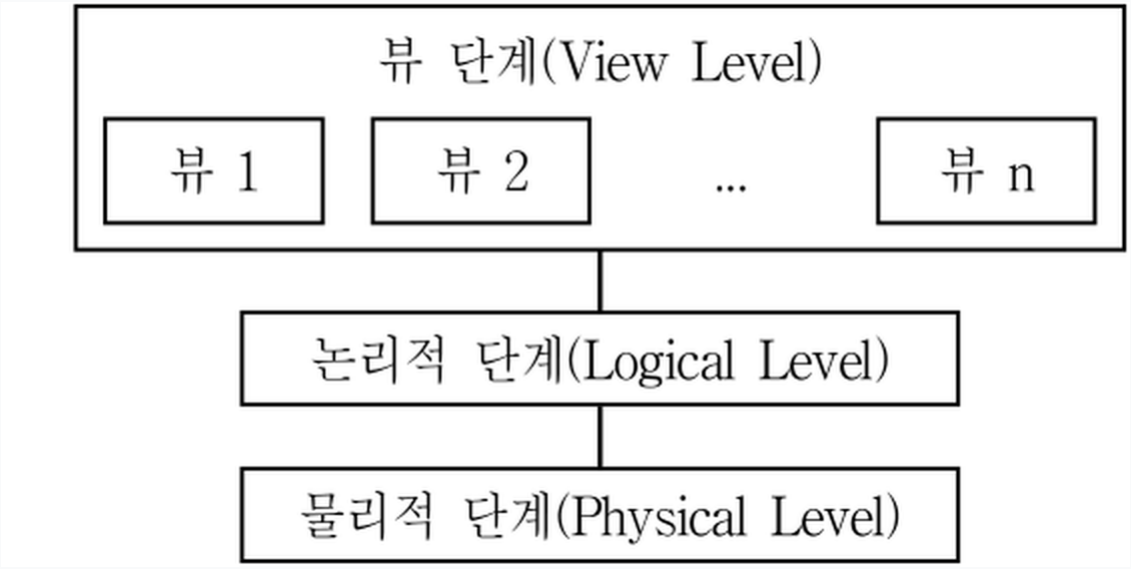
공간조인이 나오면 먼저 조인의 기준이 속성값인지 공간 관계인지 구분한다. 포함, 인접, 교차처럼 위치 관계가 기준이면 공간조인이고, 단순히 객체에 대응하는 속성을 붙이는 설명이면 일반 조인에 가깝다.

암기 포인트

공간 색인은 R-트리, 공간조인은 공간 관계, 공간 마이닝은 인접성과 분포 분석이 핵심이다.

NoteBase

문 11. 데이터 추상화는 다음 그림과 같이 3단계로 구성된다. 각 단계에 대한 설명으로 옳지 않은 것은?



① 논리적 단계에서는 데이터베이스 전반에 걸친 데이터 구조와 관계를 기술한다.

② DBA는 논리적 단계에서 어떤 데이터를 데이터베이스에 저장할지를 결정한다.

③ 물리적 단계에서는 하위 단계의 복잡한 데이터 구조들을 상세하게 기술한다.

④ 사용자들은 뷰 단계를 통하여 데이터베이스 전반에 걸친 상세 내용을 접근한다.

정답 4번

출제 포인트

이 문제는 데이터 추상화 3단계 도식에서 뷰 단계는 사용자별 부분 구조, 논리적 단계는 데이터베이스 전체의 논리 구조, 물리적 단계는 실제 저장 방법을 다룬다는 구분을 묻는다.

NoteBase

선지별 해설

① O
논리적 단계는 데이터베이스 전반에 걸친 데이터 구조와 데이터 사이의 관계를 기술한다. 전체 개념 스키마를 설명한 문장이므로 옳다.

② O
논리적 단계는 어떤 데이터를 저장할지와 그 데이터들 사이의 관계를 정하는 수준이다. DBA가 전체 논리 스키마를 설계한다는 취지이므로 옳다.

③ O
물리적 단계는 파일 구조, 레코드 배치, 인덱스처럼 실제 저장을 위한 저수준의 복잡한 데이터 구조를 상세하게 기술한다. 저장 방법을 다루는 단계라는 점에서 옳다.

④ X
뷰 단계는 각 사용자나 응용 프로그램이 필요한 일부 데이터만 보게 하는 단계이다. '데이터베이스 전반에 걸친 상세 내용'에 접근한다는 부분이 틀렸고, 올바르게는 사용자별로 필요한 부분 구조만 접근한다.

오답 포인트

뷰 단계를 최상위 단계라고 해서 전체 데이터베이스를 가장 많이 보여 주는 단계로 착각하기 쉽다. 그러나 뷰 단계의 핵심은 상세 내용을 숨기고 사용자별 필요한 부분만 보여 준다는 점이다.

다시 나오면 이렇게 판단

'무엇을 저장하고 어떤 관계인가'를 묻으면 논리적 단계, '어떻게 저장하는가'를 묻으면 물리적 단계, '사용자에게 어떤 일부만 보여 주는가'를 묻으면 뷰 단계로 판단한다.

암기 포인트

무엇과 관계는 논리, 어떻게 저장은 물리, 사용자별 일부만 보이는 것은 뷰이다.

NoteBase

문 12. 정규화에 대한 설명으로 옳지 않은 것은?

① 정규화의 목적은 각 릴레이션에 분산된 종속성을 하나의 릴레이션으로 통합하는 것이다.

② 정규화 과정을 거치지 않으면 여러 다른 종류의 정보를 하나의 릴레이션에 표현하여 그 릴레이션을 조작할 때 이상 현상이 발생할 수 있다.

③ 데이터 간에 존재하는 함수종속은 이상 현상의 원인이 될 수 있다.

④ 정규화가 잘못되면 데이터의 불필요한 중복이 발생하여 릴레이션 조작 시 문제를 유발할 수 있다.

정답 1번

출제 포인트

이 문제는 정규화의 목적과 함수종속, 이상 현상의 관계를 구분하는 문제이다.

핵심 개념

정규화는 함수종속을 기준으로 릴레이션을 분해하여 데이터 중복을 줄이고 삽입·삭제·갱신 이상을 방지하는 과정이다.
비정규화는 성능 등의 이유로 분해된 구조를 일부 통합하거나 중복을 허용하는 방향이다.
판단 기준은 통합이 아니라 분해, 중복 제거, 이상 방지라는 키워드이다.

선지별 해설

① X. 틀린 부분은 '하나의 릴레이션으로 통합'이라는 설명이다. 정규화는 여러 사실을 한곳에 합치는 과정이 아니라 함수종속을 기준으로 릴레이션을 적절히 분해하여 중복을 줄이는 과정이다. 올바르게는 '중복과 이상 현상을 줄이기 위해 릴레이션을 분해한다'이다.
② O. 정규화를 거치지 않으면 서로 다른 종류의 정보를 하나의 릴레이션에 함께 저장하게 되어 삽입 이상, 삭제 이상, 갱신 이상이 발생할 수 있다.
③ O. 데이터 간 함수종속이 부적절하게 한 릴레이션에 얽혀 있으면 특정 속성이 다른 속성에 종속되어 중복이 생기고, 이것이 이상 현상의 원인이 될 수 있다.
④ O. 정규화가 제대로 되지 않으면 불필요한 중복이 남고, 그 결과 같은 사실을 여러 곳에서 수정해야 하므로 릴레이션 조작 시 문제가 생길 수 있다.

오답 포인트

정규화와 비정규화를 '통합'과 '분해'의 방향에서 거꾸로 외우면 ①에 끌리기 쉽다. 이 문제는 정규화의 목적을 묻는데, '하나의 릴레이션으로 통합'이라는 표현은 비정규화 쪽 발상에 가깝다.

NoteBase

다시 나오면 이렇게 판단

정규화 관련 선지를 보면 먼저 중복 제거와 이상 방지인지, 아니면 통합과 중복 허용인지를 구분한다. 함수종속을 정리하려고 릴레이션을 분해하면 정규화이고, 여러 정보를 합쳐 중복을 허용하면 정규화가 아니다.

암기 포인트

정규화는 분해로 중복을 줄이고, 비정규화는 통합으로 성능을 보완한다.

NoteBase

문 13. 두 트랜잭션 T_1, T_2가 다음과 같은 트랜잭션 스케줄로 실행될 때, 발생되는 충돌(conflict)로 옳은 것은? (단, R은 읽기연산, W는 쓰기연산에 해당하며, $R_1(A)$는 "트랜잭션 T_1이 A를 읽는다", $W_2(B)$는 "트랜잭션 T_2가 B에 값을 쓴다"는 의미이다)

$R_1(A)$; $W_1(A)$; $R_2(A)$; $W_2(A)$; $R_2(B)$; $W_2(B)$; commit_2; $R_1(B)$; $W_1(B)$; commit_1;

① 충돌이 발생하지 않음

② 읽기-쓰기 충돌(read-write conflict)

③ 읽기-읽기 충돌(read-read conflict)

④ 쓰기-쓰기 충돌(write-write conflict)

정답 2번

출제 포인트

이 문제는 트랜잭션 스케줄에서 같은 데이터 항목을 서로 다른 트랜잭션이 접근할 때, 적어도 한쪽이 쓰기이면 충돌이 된다는 기준을 적용할 수 있는지를 묻는 문제이다.

핵심 개념

충돌은 서로 다른 두 트랜잭션이 같은 데이터 항목을 접근하고 그중 적어도 하나가 쓰기일 때 발생한다.
읽기-읽기는 충돌이 아니고, 읽기-쓰기나 쓰기-쓰기 조합은 모두 충돌이 된다.

선지별 해설

① X: A에 대해 $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$가, B에 대해 $R_2(B)$, $W_2(B)$, $R_1(B)$, $W_1(B)$가 수행되므로 같은 항목에 대한 교차 접근이 존재한다. 따라서 충돌이 발생하지 않는다는 진술은 틀리다. 올바르게는 같은 데이터 항목에 대한 충돌이 실제로 발생한다.
② O: $R_1(A)$ 뒤의 $W_2(A)$, $R_2(B)$ 뒤의 $W_1(B)$처럼 한 트랜잭션이 읽은 항목을 다른 트랜잭션이 쓰는 경우가 있으므로 읽기-쓰기 충돌이 발생한다. 공식 정답은 2번으로 처리되어 있다.
③ X: 읽기-읽기는 두 트랜잭션이 같은 항목을 읽기만 하는 경우인데(예: $R_1(A)$와 $R_2(A)$), 이때는 데이터 값이 변경되지 않으므로 충돌로 보지 않는다. 올바르게는 적어도 한 연산이 쓰기여야 충돌이다.
④ △: 쓰기-쓰기 충돌은 서로 다른 두 트랜잭션이 같은 항목에 모두 쓰기를 수행하면 성립한다. 이 스케줄에서도 A에 대해 $W_1(A)$와 $W_2(A)$가, B에 대해 $W_2(B)$와 $W_1(B)$가 나타나므로 쓰기-쓰기 충돌 역시 실제로 존재한다. 즉 이 선지의 진술 자체도 성립한다. 공식 정답이 2번(읽기-쓰기 충돌)으로 처리된 것은 읽기와 쓰기가 교차하는 충돌을 대표로 물은 것으로 이해되나, 쓰기-쓰기 충돌도 분명히 발생하므로 2번과 4번이 함께 성립하는 문항으로 보인다(정답 수동 확인 필요).

NoteBase

오답 포인트

엄밀하게 충돌쌍을 세면 읽기-쓰기 충돌(예: $R_1(A)$와 $W_2(A)$, $W_2(B)$와 $R_1(B)$)뿐 아니라 쓰기-쓰기 충돌($W_1(A)$와 $W_2(A)$, $W_2(B)$와 $W_1(B)$)도 실제로 나타난다. 즉 2번과 4번이 모두 성립하는 셈이어서, 공식 정답은 2번이지만 4번도 배제하기 어렵다(정답 수동 확인 필요). 다만 3번처럼 읽기-읽기를 충돌로 보는 것은 기본 정의상 분명히 틀린 판단이다.

다시 나오면 이렇게 판단

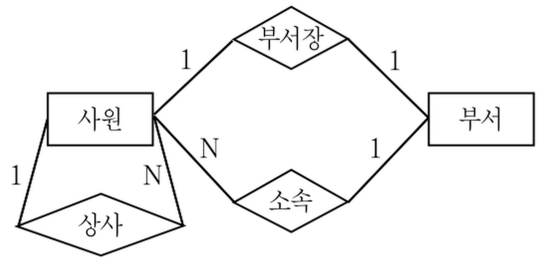
먼저 같은 데이터 항목인지 확인하고, 그다음 서로 다른 트랜잭션의 연산인지 본다. 마지막으로 두 연산 중 하나라도 쓰기이면 충돌로 판단하고, 둘 다 읽기이면 충돌이 아니라고 정리하면 된다.

암기 포인트

충돌 판단의 핵심은 같은 항목, 다른 트랜잭션, 적어도 한 번의 쓰기이다.

NoteBase

문 14. 다음과 같이 사원과 부서 테이블을 이용하는 회사가 직원의 급여를 인상할 때 오류를 방지하기 위해 <설명>과 같은 SQL99 표준에 따른 트리거를 정의하여 사용 중이다. 이와 같은 조건에서 직원들의 급여 인상을 위한 <보기>의 질의를 수행하였다. 급여 인상의 결과를 보기 위해 질의 "SELECT 사번, 이름, 급여 FROM 사원"을 수행한 결과로 옳은 것은?



사원

사번	이름	소속	상사	급여
100	홍길동	10	NULL	500
200	이몽룡	20	100	400
300	성춘향	20	200	300
400	박대감	30	100	350

부서

부서번호	부서명	부서장
10	사장실	100
20	총무부	200
30	영업부	400

<설 명>

```
CREATE TRIGGER Employment
AFTER UPDATE OF 급여 ON 사원
REFERENCING OLD ROW AS O, NEW ROW AS N
FOR EACH ROW
WHEN (N.급여 > O.급여 * 1.1)
    UPDATE 사원
    SET 급여 = O.급여 * 1.1
    WHERE 사원.사번 = O.사번;
```

NoteBase

<보 기>

```
UPDATE 사원 SET 급여 = 급여 * 1.2
WHERE 사번 IN (SELECT 부서장 FROM 부서);
UPDATE 사원 SET 급여 = 급여 * 1.05
WHERE 사번 != ALL (SELECT 부서장 FROM 부서);
UPDATE 사원 SET 급여 = 급여 * 1.1 WHERE 사번 = 100;
```

① 사원

사번	이름	급여
100	홍길동	550
200	이몽룡	420
300	성춘향	315
400	박대감	380

② 사원

사번	이름	급여
100	홍길동	550
200	이몽룡	420
300	성춘향	330
400	박대감	380

③ 사원

사번	이름	급여
100	홍길동	605
200	이몽룡	440
300	성춘향	315
400	박대감	385

NoteBase

④

사원		
사번	이름	급여
100	홍길동	605
200	이몽룡	440
300	성춘향	330
400	박대감	385

정답 3번

출제 포인트

AFTER UPDATE 행 트리거가 급여 인상률을 10% 초과할 때만 OLD 급여 기준 10% 인상으로 되돌리는 동작과, 여러 UPDATE 문이 순차적으로 누적 적용되는지를 묻는 문제이다.

사례 정리

초기 부서장은 100, 200, 400이다.
트리거는 각 행의 급여가 갱신된 뒤 NEW.급여 > OLD.급여 * 1.1 이면 그 행의 급여를 OLD.급여 * 1.1로 다시 바꾼다.
따라서 한 번의 UPDATE에서 허용되는 최대 인상폭은 10%이고, 다음 UPDATE는 직전 결과를 기준으로 다시 계산된다.

핵심 개념

이 트리거는 행 단위 AFTER UPDATE 트리거이다. 각 사원 행마다 OLD 값과 NEW 값을 비교해 10%를 초과한 인상만 제한한다. 조건식이 > 이므로 정확히 10% 인상인 경우는 트리거가 실행되지 않는다.

적용 기준

첫 번째 UPDATE는 부서장 100, 200, 400을 20% 인상하려 하지만 모두 트리거에 걸려 각각 500→550, 400→440, 350→385가 된다.
두 번째 UPDATE의 조건인 사번 != ALL (SELECT 부서장 FROM 부서)는 부서장이 아닌 사원만 뜻하므로 300만 대상이다. 300은 300→315가 되며 10% 초과가 아니어서 그대로 유지된다.
세 번째 UPDATE는 100을 550→605로 바꾸는데 이는 정확히 10% 인상이다. 조건이 > 이므로 트리거가 작동하지 않아 605가 유지된다.

NoteBase

선지별 해설

① X
홍길동의 급여를 550으로 본 것이 틀렸다. 마지막 UPDATE에서 100번 사원은 현재 급여 550의 1.1배인 605가 되고, 이는 정확히 10% 인상이어서 다시 조정되지 않는다. 올바르게는 605이다.
이몽룡을 420, 박대감을 380으로 본 것도 틀렸다. 첫 번째 UPDATE에서 부서장 200, 400은 각각 440, 385로 제한된다.

② X
성춘향을 330으로 본 것이 틀렸다. 300번 사원은 두 번째 UPDATE에서만 5% 인상되어 300→315가 된다. 10% 초과가 아니므로 트리거가 작동하지 않는다. 올바르게는 315이다.
홍길동 550, 이몽룡 420, 박대감 380으로 본 부분도 틀렸고, 올바르게는 각각 605, 440, 385이다.

③ O
첫 번째 UPDATE 뒤 부서장 100, 200, 400의 급여는 각각 550, 440, 385가 된다.
두 번째 UPDATE는 부서장이 아닌 300만 대상으로 하므로 315가 된다.
세 번째 UPDATE는 100을 605로 만들고, 이는 정확히 10% 인상이라 트리거 조건을 만족하지 않아 그대로 유지된다.

④ X
성춘향을 330으로 본 것이 틀렸다. 두 번째 UPDATE는 5% 인상만 적용되므로 300번 사원의 최종 급여는 315이다. 올바르게는 315이다.
나머지 100=605, 200=440, 400=385는 맞다.

오답 포인트

함정은 두 가지이다. 첫째, 트리거 조건을 >= 로 착각하면 마지막 550→605도 다시 막힌다고 오해한다. 둘째, 두 번째 UPDATE에서 300의 5% 인상까지 10% 상한에 걸린다고 착각하면 315 대신 330을 고르게 된다. 또한 사번 != ALL (...)을 부서장 제외 조건으로 읽지 못하면 대상 행을 잘못 잡게 된다.

다시 나오면 이렇게 판단

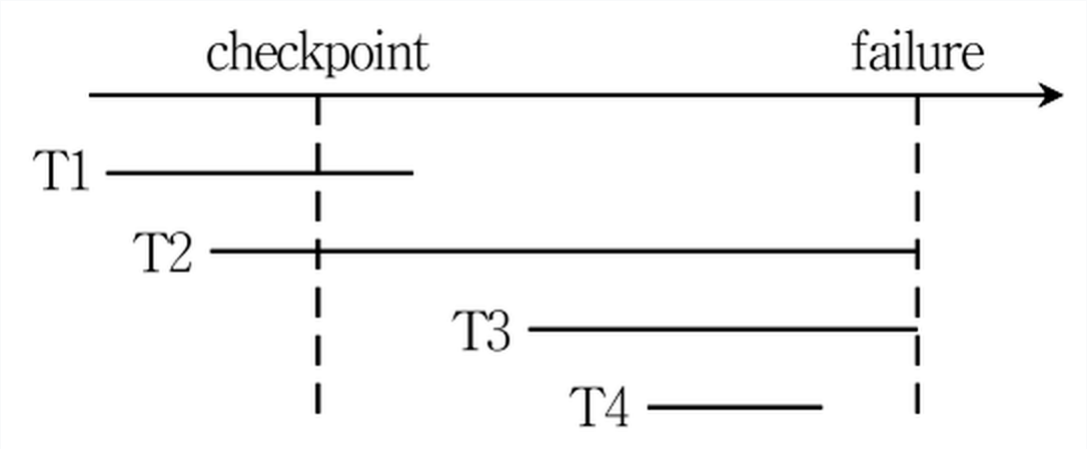
트리거 적용 문제는 먼저 UPDATE 대상 행을 찾고, 각 행마다 OLD와 NEW를 비교한 뒤, 트리거가 값을 다시 바꾸는지 확인한다. 그 다음 문장은 앞 문장에서 이미 조정된 결과를 새로운 OLD 값으로 삼아 순서대로 누적 계산하면 된다. 마지막으로 비교 연산자가 > 인지 >= 인지까지 확인해야 한다.

암기 포인트

행 AFTER UPDATE 트리거는 각 행의 OLD와 NEW를 비교하고, 다음 UPDATE는 직전 결과를 다시 기준값으로 사용한다.

NoteBase

문 15. 검사점(checkpoint) 회복 기법과 관련하여 트랜잭션의 실행 상태가 다음 그림과 같다고 하자. 데이터 A, B, C, D의 초기값이 각각 100, 200, 100, 200일 때 T1은 A에 100을 더하고, T2는 B에서 100을 빼고, T3은 C에 100을 더하고, T4는 D에서 100을 뺀다고 하자. 회복이 수행된 후 A, B, C, D의 값으로 옳은 것은?



① A: 100, B: 200, C: 100, D: 200

② A: 200, B: 100, C: 100, D: 200

③ A: 200, B: 200, C: 100, D: 100

④ A: 200, B: 100, C: 200, D: 100

정답 3번

출제 포인트

체크포인트 회복에서 장애 시점 미완료 트랜잭션은 UNDO하고, 마지막 체크포인트 이후 완료된 트랜잭션은 REDO하는 기준을 묻는 문제이다.

NoteBase

선지별 해설

① X
T1은 체크포인트 시점에 진행 중이었지만 장애 전 완료되었으므로 REDO 대상이라 A는 200이 된다. 선지의 A=100이 틀렸고, 올바르게는 A=200이다.
T2와 T3는 장애 시점 미완료이므로 UNDO되어 B=200, C=100은 맞다.
T4는 체크포인트 이후 완료되었으므로 REDO 대상이라 D는 100이 된다. 선지의 D=200이 틀렸고, 올바르게는 D=100이다.

② X
T1은 장애 전 완료되어 REDO되므로 A=200은 맞다.
T2는 장애 시점 미완료이므로 UNDO되어 B는 200이어야 한다. 선지의 B=100이 틀렸고, 올바르게는 B=200이다.
T3는 미완료이므로 UNDO되어 C=100은 맞다.
T4는 체크포인트 이후 완료되었으므로 REDO되어 D는 100이어야 한다. 선지의 D=200이 틀렸고, 올바르게는 D=100이다.

③ O
T1은 체크포인트 시점에 진행 중이었지만 장애 전 완료되었으므로 REDO되어 A=200이다.
T2는 장애 시점까지 끝나지 않아 UNDO되므로 B=200이다.
T3는 체크포인트 이후 시작했지만 끝나지 않아 UNDO되므로 C=100이다.
T4는 체크포인트 이후 시작해서 장애 전 완료되었으므로 REDO되어 D=100이다.

④ X
T1의 REDO로 A=200은 맞다.
T2는 미완료라 UNDO되어 B=100이 아니라 200이다. 선지의 B=100이 틀렸고, 올바르게는 B=200이다.
T3도 미완료라 UNDO되어 C=200이 아니라 100이다. 선지의 C=200이 틀렸고, 올바르게는 C=100이다.
T4는 완료되어 REDO되므로 D=100은 맞다.

오답 포인트

체크포인트 이전에 시작한 T1을 반영 대상에서 빼는 실수가 잦다. 회복 판단 기준은 시작 시점 자체가 아니라 장애 시점 완료 여부와 체크포인트 이후 완료 여부이다.

다시 나오면 이렇게 판단

먼저 장애 시점에 완료된 트랜잭션과 미완료 트랜잭션을 나눈다. 그다음 완료된 것은 REDO, 미완료인 것은 UNDO로 분류한 뒤 각 데이터 항목 값에 반영하면 된다.

NoteBase

암기 포인트

체크포인트 회복은 장애 전 완료는 REDO, 장애 시점 진행 중은 UNDO로 정리된다.

NoteBase

문 16. 고객들 중 꽃을 한번도 구매하지 않은 고객의 고객ID, 이름, 주소를 다음 데이터베이스로부터 검색하려고 할 때 옳은 것은? (단, PK는 PRIMARY KEY, FK는 FOREIGN KEY를 의미함)

꽃

꽃ID(PK)	꽃이름	가격
11	Spray Roses	19
22	Sunflower	44
23	Love's First Bloom	35

고객

고객ID(PK)	이름	주소
1010	S. Brass	Pittsburgh, PA
3005	Romeo	Lexington, KY
2002	Juliet	Norman, OK

주문

주문번호(PK)	꽃ID(FK)	고객ID(FK)	수량	판매날짜
23010	11	1010	4	2013-08-22
23011	11	3005	3	2013-09-11
23012	11	1010	1	2013-10-12
23013	23	1010	3	2013-08-01

①
```
SELECT C.고객ID, C.이름, C.주소
FROM 고객 C
WHERE C.고객ID NOT IN
(SELECT C.고객ID FROM 주문 O WHERE C.고객ID = O.고객ID);
```

②
```
SELECT C.고객ID, C.이름, C.주소
FROM 주문 O LEFT OUTER JOIN 고객 C ON O.고객ID = C.고객ID
WHERE  O.주문번호 IS NULL;
```

③
```
SELECT C.고객ID, C.이름, C.주소
FROM 주문 O, 고객 C
WHERE O.고객ID = C.고객ID AND O.주문번호 IS NULL;
```

④
```
SELECT C.고객ID, C.이름, C.주소
FROM 고객 C
WHERE EXISTS
(SELECT * FROM 주문 O WHERE C.고객ID = O.고객ID);
```

정답 1번

출제 포인트

고객 테이블 전체에서 주문 테이블에 한 번도 나타나지 않은 고객ID를 골라내는 SQL 패턴을 묻는 문제이다.

자료 읽는 순서

1. 고객 테이블의 고객ID를 확인하면 1010, 3005, 2002이다.
2. 주문 테이블의 고객ID(FK)를 확인하면 1010과 3005만 나타난다.
3. 따라서 주문 테이블에 한 번도 나타나지 않은 고객 2002를 반환하는 구문이 정답이다.

NoteBase

핵심 근거

고객 테이블 기준 전체 고객은 1010, 3005, 2002이다.
주문 테이블의 고객ID 열에는 1010과 3005만 있으므로 주문 이력이 없는 고객은 2002 Juliet 한 명뿐이다.

선지별 해설

① O
근거 위치: 고객 테이블의 고객ID 열 전체와 주문 테이블의 고객ID(FK) 열이다.
상관 서브쿼리로 현재 고객 C의 고객ID와 같은 주문이 있는지를 주문 테이블에서 찾고, 그 결과가 없으면 NOT IN 조건이 참이 된다.
따라서 주문이 없는 고객 2002만 남으므로 정답 조건에 맞는다.

② X
근거 위치: 주문 테이블의 주문번호(PK) 열이다.
틀린 표현은 `FROM 주문 O LEFT OUTER JOIN 고객 C ... WHERE O.주문번호 IS NULL` 부분이다. LEFT JOIN의 기준 테이블이 주문 O이므로 결과 행마다 O의 주문번호는 원래 존재하며 NULL이 될 수 없다.
올바르게는 `FROM 고객 C LEFT OUTER JOIN 주문 O ON C.고객ID = O.고객ID WHERE O.주문번호 IS NULL`처럼 고객을 기준으로 조인해야 미주문 고객을 찾을 수 있다.

③ X
근거 위치: 주문 테이블의 주문번호(PK) 열과 조인 조건 `O.고객ID = C.고객ID`이다.
틀린 표현은 내부 조인 상태에서 `O.주문번호 IS NULL`을 검사하는 부분이다. 주문번호는 기본키이므로 NULL이 아니고, 내부 조인은 주문이 있는 고객만 남기므로 결과가 공집합이 된다.
올바르게는 내부 조인이 아니라 고객 기준 외부 조인 후 주문 쪽 NULL을 검사하거나 `NOT EXISTS`를 사용해야 한다.

④ X
근거 위치: 주문 테이블의 고객ID(FK) 열이다.
틀린 표현은 `WHERE EXISTS` 부분이다. EXISTS는 해당 고객의 주문이 하나라도 있으면 참이 되므로 1010과 3005처럼 이미 주문한 고객을 찾는다.
올바르게는 `WHERE NOT EXISTS (SELECT * FROM 주문 O WHERE C.고객ID = O.고객ID)`처럼 부정을 사용해야 주문이 없는 고객을 찾을 수 있다.

오답 포인트

이 문제의 함정은 두 가지이다. 첫째, 꽃ID 22가 주문에 없다는 사실에 끌려 꽃이 안 팔린 경우와 고객이 한 번도 구매하지 않은 경우를 혼동하기 쉽다. 문제는 꽃이 아니라 고객을 찾는 것이다. 둘째, OUTER JOIN을 쓰더라도 기준 테이블을 주문으로 잡으면 NULL 검사가 성립하지 않는다는 점을 놓치기 쉽다.

NoteBase

다시 나오면 이렇게 판단

미구매 고객 문제는 먼저 고객 전체 집합에서 주문 테이블에 등장한 고객 집합을 뺀다고 생각하면 된다. SQL은 `NOT EXISTS`, `NOT IN`, 또는 `고객 LEFT JOIN 주문 후 주문쪽 NULL` 중 하나로 구현되며, 반드시 고객 테이블이 기준이 되어야 한다.

암기 포인트

미구매 고객은 고객 기준으로 잡고 주문 존재 여부를 부정해서 찾는다.

NoteBase

문 17. 트랜잭션의 교착상태 방지(deadlock prevention) 프로토콜 중에서 <보기>의 설명을 만족하는 프로토콜은?

<보기>
◯ 타임스탬프를 필요로 하지 않음
◯ T_i가 T_j에서 로크를 건 항목 X가 필요할 경우, T_j가 블로킹된 상태가 아니면(로크가 걸린 다른 항목을 기다리는 상태가 아니면) T_i가 블로킹이 되어 대기를 시작하고, T_j가 블로킹이 된 상태이면 T_i는 철회(abort)됨

① 무 대기(no waiting)
② 신중 대기(cautious waiting)
③ 상처-대기(wound-wait)
④ 대기-죽음(wait-die)

정답 2번

출제 포인트

이 문제는 교착상태 방지 프로토콜인 no waiting, cautious waiting, wound-wait, wait-die의 구별 기준을 묻는 문제이다.

핵심 개념

cautious waiting은 잠금을 가진 트랜잭션이 현재 다른 자원을 기다리며 블로킹된 상태인지 아닌지를 기준으로 대기 여부를 정하는 교착상태 방지 기법이다.
no waiting은 충돌이 나면 기다리지 않고 즉시 철회시키는 방식이다.
wound-wait와 wait-die는 모두 타임스탬프가 필요한 방식이며, 오래된 트랜잭션과 새로운 트랜잭션 사이의 우선순위로 대기와 철회를 결정한다.

NoteBase

선지별 해설

① X
무 대기(no waiting)는 잠긴 항목을 요청했을 때 대기를 허용하지 않고 즉시 abort시키는 방식이다. 보기처럼 상대가 블로킹되지 않았으면 기다리고, 블로킹되었으면 abort시키는 방식이 아니다.
틀린 표현은 '조건부로 대기한다'는 취지이다. 올바르게는 '충돌이 발생하면 기다리지 않고 즉시 철회한다'이다.

② O
신중 대기(cautious waiting)는 타임스탬프를 사용하지 않으며, T_i가 T_j가 잠근 항목을 요청할 때 T_j가 아직 다른 항목을 기다리지 않는 상태이면 T_i를 기다리게 하고, T_j가 이미 블로킹 상태이면 T_i를 abort시킨다. 보기의 두 조건과 정확히 일치한다.

③ X
상처-대기(wound-wait)는 타임스탬프 기반 방식이다. 오래된 트랜잭션이 새로운 트랜잭션이 가진 잠금을 요구하면 새로운 쪽을 abort시키고, 반대로 새로운 트랜잭션은 오래된 쪽을 기다린다. 보기처럼 '상대가 블로킹 상태인지'를 기준으로 판단하지 않는다.
틀린 표현은 '타임스탬프가 필요하지 않음'과 '상대의 블로킹 여부로 결정함'이다. 올바르게는 '타임스탬프가 필요하며 연령 우선순위로 대기와 철회를 정한다'이다.

④ X
대기-죽음(wait-die)도 타임스탬프 기반 방식이다. 오래된 트랜잭션은 기다릴 수 있지만, 새로운 트랜잭션이 오래된 트랜잭션이 가진 잠금을 요구하면 새로운 쪽이 abort된다. 보기와 달리 상대가 블로킹되었는지가 기준이 아니다.
틀린 표현은 '블로킹 여부에 따라 wait 또는 die를 정한다'는 취지이다. 올바르게는 '트랜잭션의 타임스탬프 선후관계로 wait 또는 die를 정한다'이다.

오답 포인트

③과 ④는 둘 다 교착상태 방지 기법이라는 점은 같지만, 둘 다 타임스탬프 기반이라는 공통점이 있다. 보기의 첫 문장에 '타임스탬프를 필요로 하지 않음'이 명시되어 있으므로 ③, ④는 먼저 배제된다. ①도 비타임스탬프 방식이지만 '절대 대기하지 않음'이라는 점에서 보기의 조건부 대기와 다르다.

다시 나오면 이렇게 판단

먼저 타임스탬프 필요 여부로 프로토콜을 둘로 나눈다. 타임스탬프가 필요하면 wound-wait 또는 wait-die이고, 필요하지 않으면 no waiting 또는 cautious waiting이다. 그다음 대기를 전혀 허용하지 않으면 no waiting, 상대가 블로킹되지 않았을 때만 기다리게 하면 cautious waiting이다.

NoteBase

암기 포인트

no waiting은 무조건 abort, cautious waiting은 상대가 안 막혀 있으면 wait, wound-wait와 wait-die는 타임스탬프 기반이다.

NoteBase

문 18. 릴레이션 R(A, B, C, D, E)에 대한 종속성의 집합 F = {A→C, AC→D, D→C, D→E}가 주어졌을 때, 종속성 집합 F의 최소 집합(minimal set)으로 옳은 것은?

① {A→C, A→D, D→C, D→E}

② {A→C, C→D, D→E}

③ {A→D, D→C, D→E}

④ {A→C, A→D}

정답 3번

출제 포인트

이 문제는 함수 종속성 집합의 최소 집합, 즉 원래 집합과 동치이면서 왼쪽의 불필요한 속성과 중복 함수 종속을 제거한 형태를 판별하는 문제이다.

핵심 개념

최소 집합은 원래 함수 종속성 집합과 동치여야 하고, 각 함수 종속의 왼쪽에는 불필요한 속성이 없어야 하며, 어떤 함수 종속도 다른 종속들로 대체 가능하면 제거되어야 한다.
이 문제에서는 $AC \rightarrow D$에서 C가 불필요하다. $A \rightarrow C$가 있으므로 A만으로 C를 얻고 다시 $AC \rightarrow D$를 적용할 수 있어 $A \rightarrow D$로 줄어든다.
그다음 $A \rightarrow C$는 $A \rightarrow D$와 $D \rightarrow C$로 유도되므로 중복이다. 따라서 최소 집합은 $\{A \rightarrow D, D \rightarrow C, D \rightarrow E\}$이다.

NoteBase

선지별 해설

① X
- 이 집합은 원래 집합과 동치는 될 수 있지만 최소 집합은 아니다.
- 틀린 부분은 $A \to C$를 그대로 남겨 둔 점이다. $A \to D$와 $D \to C$가 있으면 $A \to C$는 이미 유도되므로 중복이다.
- 올바르게는 $A \to C$를 제거한 $\{A \to D, D \to C, D \to E\}$가 최소 집합이다.

② X
- 틀린 부분은 $C \to D$이다. 원래 집합에서는 C만으로 D를 결정할 수 없다.
- 또한 $D \to C$가 빠져 있어 원래 집합과 동치도 아니다. $D \to C$가 성립한다고 해서 역으로 $C \to D$가 성립하는 것은 아니다.
- 올바르게는 $C \to D$가 아니라 $A \to D$와 $D \to C$가 필요하며, 최소 집합은 $\{A \to D, D \to C, D \to E\}$이다.

③ O
- $AC \to D$에서 C는 $A \to C$ 때문에 불필요하므로 $A \to D$로 축약된다.
- 이어서 $A \to C$는 $A \to D$와 $D \to C$로 유도되므로 제거된다.
- 따라서 $\{A \to D, D \to C, D \to E\}$는 원래 집합과 동치이면서 더 줄일 수 없는 최소 집합이다.

④ X
- 틀린 부분은 $D \to C, D \to E$를 빠뜨린 점이다.
- 이 두 함수 종속은 다른 종속들만으로 유도되지 않으므로 집합 $\{A \to C, A \to D\}$는 원래 집합과 동치가 아니다.
- 올바르게는 $A \to D, D \to C, D \to E$를 모두 포함해야 한다.

오답 포인트

최소 집합 문제에서는 원래 집합에 있던 함수 종속을 그대로 남겨 두기 쉽고, 특히 $D \to C$를 보고 역함수 종속처럼 $C \to D$로 바꾸는 실수를 하기 쉽다. 이 문항의 함정은 동치성과 최소성 두 조건을 동시에 보지 않으면 ①이나 ②로 흔들린다는 점이다.

다시 나오면 이렇게 판단

최소 집합은 먼저 왼쪽의 불필요한 속성을 지우고, 다음으로 다른 함수 종속들로 유도되는 중복 종속을 제거한 뒤, 마지막에 원래 집합과 동치인지 확인하면서 판단한다. 이 문항에서는 $AC \to D$를 $A \to D$로 줄인 뒤 $A \to C$를 제거하면 바로 정답이 나온다.

암기 포인트

최소 커버는 왼쪽 축약, 중복 제거, 동치 확인의 순서로 정리된다. 이 문항은 $AC \to D$를 $A \to D$로 바꾸고 $A \to C$를 버리면 끝이다.

NoteBase

문 19. 학과_등록 테이블이 다음과 같을 때, 학과_등록 테이블에 대한 설명으로 옳지 않은 것은? (단, 밑줄은 기본키이다)

학과_등록

학번	과목번호	학점	과목이름
100	A01	A	JAVA
101	B01	D	C#
101	B03	A	DB 시스템
100	B01	B	C#
200	A02	C	Direct X

① 과목이름을 수정하려 할 때 갱신이상이 발생할 수 있다.

② 제2 정규형을 만족시킨다.

③ 학번이 200인 학생이 수강을 철회하면 삭제이상이 발생한다.

④ 아직 수강생이 없는 새로운 과목을 삽입할 때, 삽입이상이 발생한다.

정답 2번

출제 포인트

복합 기본키를 가진 릴레이션에서 부분 함수 종속이 있으면 제2정규형을 만족하지 못하고, 그에 따라 갱신 이상·삽입 이상·삭제 이상이 발생할 수 있음을 묻는 문제이다.

핵심 개념

제2정규형은 제1정규형을 만족하면서, 복합키가 있을 때 모든 일반 속성이 기본키 전체에 완전 함수 종속되는 상태이다.
이 릴레이션의 기본키는 (학번, 과목번호)인데, 과목이름은 기본키 전체가 아니라 과목번호만으로 결정된다. 즉 과목번호 -> 과목이름이 성립하므로 부분 함수 종속이 존재하고, 따라서 제2정규형을 만족하지 않는다.

NoteBase

선지별 해설

① O
과목번호 B01의 과목이름이 두 행에서 모두 C#으로 반복된다. 과목이름을 수정하려면 관련된 모든 행을 함께 바꾸어야 하므로 갱신 이상이 발생할 수 있다.

② X
틀린 부분은 "제2 정규형을 만족시킨다"이다.
실제로는 과목번호 -> 과목이름이라는 부분 함수 종속이 있으므로 제2정규형을 만족하지 않는다.

③ O
학번 200의 수강 정보가 A02 한 건뿐이므로 이 행을 삭제하면 학생 200의 수강 기록과 과목 A02의 과목이름 Direct X 정보가 함께 사라질 수 있다. 이는 삭제 이상에 해당한다.

④ O
아직 수강생이 없는 새 과목은 현재 구조만으로는 학번과 학점 없이 따로 저장하기 어렵다. 과목번호와 과목이름만 삽입하지 못하는 상황이 생기므로 삽입 이상이 발생한다.

오답 포인트

제2정규형을 단순히 반복 속성의 유무로만 판단하면 ②를 맞히기 어렵다. 이 문항의 함정은 기본키가 복합키라는 점과, 과목이름이 그 전체가 아니라 일부인 과목번호에만 종속된다는 점이다.

다시 나오면 이렇게 판단

복합키가 보이면 먼저 일반 속성이 키 전체에 종속되는지 확인한다. 학번이나 과목번호 같은 키의 일부만으로 결정되는 속성이 있으면 부분 함수 종속이므로 제2정규형 위반이라고 판단할 수 있다.

암기 포인트

복합키의 일부 -> 일반 속성이면 부분 함수 종속이고, 제2정규형을 만족하지 않는다.

NoteBase

문 20. 다음은 데이터로부터의 지식발견(KDD: Knowledge Discovery from Data)을 위한 단계별 과정이다. 빈칸에 들어갈 말로 옳은 것은?



<데이터로부터의 지식발견을 위한 단계별 과정>

① ㄱ: 데이터 통합, ㄴ: 데이터마이닝, ㄷ: 패턴평가 및 표현

② ㄱ: 데이터마이닝, ㄴ: 데이터 통합, ㄷ: 패턴평가 및 표현

③ ㄱ: 데이터 통합, ㄴ: 패턴평가 및 표현, ㄷ: 데이터마이닝

④ ㄱ: 패턴평가 및 표현, ㄴ: 데이터마이닝, ㄷ: 데이터 통합

정답 1번

출제 포인트

이 문제는 KDD의 표준 처리 순서에서 데이터 정제 이후와 데이터 변환 이후에 어떤 단계가 오는지를 묻는 문제이다.

올바른 순서

1. 데이터 정제 - 잡음과 불일치 데이터를 정리하는 단계이다.
2. 데이터 통합 - 여러 데이터 원천을 하나로 결합하는 단계이다.
3. 데이터 선택 - 분석에 필요한 데이터만 추리는 단계이다.
4. 데이터 변환 - 마이닝에 맞게 정규화, 요약, 형식 변환을 하는 단계이다.
5. 데이터마이닝 - 규칙과 패턴을 실제로 추출하는 단계이다.
6. 패턴평가 및 표현 - 추출된 패턴의 유의성을 평가하고 지식 형태로 표현하는 단계이다.

왜 이 순서인가: 여러 원천을 먼저 통합해야 필요한 데이터를 선택할 수 있고, 변환된 데이터가 데이터마이닝의 입력이 되며, 마이닝 결과가 나온 뒤에야 패턴평가 및 표현이 가능하다.

NoteBase

선지별 해설

① O: ㄱ=데이터 통합, ㄴ=데이터마이닝, ㄷ=패턴평가 및 표현으로 배치되어 KDD의 표준 순서와 일치한다.
② X: 틀린 부분은 ㄱ을 데이터마이닝, ㄴ을 데이터 통합으로 바꾼 점이다. 올바르게는 데이터 정제 다음이 데이터 통합이고, 데이터 변환 다음이 데이터마이닝이다.
③ X: 틀린 부분은 ㄴ을 패턴평가 및 표현, ㄷ을 데이터마이닝으로 바꾼 점이다. 올바르게는 데이터 변환 뒤에는 데이터마이닝이 오고, 패턴평가 및 표현은 그 다음 마지막 단계이다.
④ X: 틀린 부분은 ㄱ을 패턴평가 및 표현, ㄷ을 데이터 통합으로 둔 점이다. 올바르게는 패턴평가 및 표현은 가장 마지막 단계이고, 데이터 통합은 데이터 선택 이전 단계이다.

오답 포인트

데이터 통합과 데이터 선택을 모두 전처리 단계로만 기억하면 두 위치를 섞기 쉽다. 또한 패턴평가 및 표현은 데이터마이닝의 결과를 대상으로 하는 단계이므로, 데이터마이닝보다 앞에 올 수 없다는 점이 핵심 함정이다.

다시 나오면 이렇게 판단

KDD는 전처리 단계인 정제-통합-선택-변환을 먼저 놓고, 그 다음 분석 단계인 데이터마이닝, 마지막으로 결과 정리 단계인 패턴평가 및 표현을 배치하면 된다.

암기 포인트

KDD 흐름은 정제-통합-선택-변환-마이닝-평가 및 표현의 순서이다.

NoteBase